Write a bare-bones Markov text generator.

Implement a function of the form
`finish_sentence(sentence, n, corpus, deterministic=False)`
that takes four arguments:

1. a `sentence` [list of tokens] that we're trying to build on,
2. `n` [int], the length of n-grams to use for prediction, and
3. a source `corpus` [list of tokens]
4. a flag indicating whether the process should be `deterministic` [bool]

and returns an extended sentence until the first `.`, `?`, or `!` is found OR until it has 10 total tokens.

If the input flag `deterministic` is true, choose at each step the single most probable next token. When two tokens are equally probable, choose the one that occurs first in the corpus.

If `deterministic` is false, draw the next word randomly from the appropriate distribution. Use stupid backoff and no smoothing.

Provide some example applications of your function in both deterministic and stochastic modes, for a few sets of seed words and a few different n.

As one (simple) test case, use the following inputs:

```
sentence = ['she', 'was', 'not']
n = 3
nltk.word_tokenize(nltk.corpus.gutenberg.raw('austen-sense.txt').lower())
deterministic = True
```

The result should be
`['she', 'was', 'not', 'in', 'the', 'world', '.']`
OR
`['she', 'was', 'not', 'in', 'the', 'world', ',', 'and', 'the', 'two']`

Add your method to a file/module named `mtg.py` and use the test script `text_mtg.py` to verify that this example works.

You should turn in a document (`.txt`, `.md`, or `.pdf`) answering all of the red items above. You should also turn in Python scripts (`.py`) for *each* of the blue items.